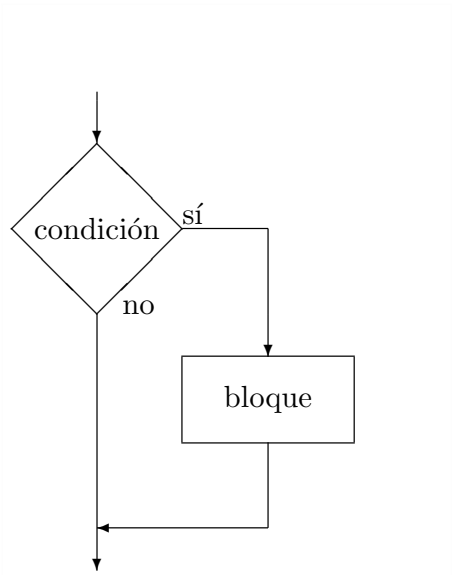
FUNDAMENTOS DE COMPUTACIÓN
Grado en ingeniería en tecnologías Industriales
Universidad de Cantabria

Práctica 5 (laboratorio)
Octubre 2019

Bifurcaciones condicionales

- **La estrucutura de control** if ... La programación resultaría demasiado rígida si tuviéramos que restringirnos a enumerar una sucesión de instrucciones que se ejecutaran una detrás de otra. A veces, queremos *seguir caminos distintos* en función de alguna circunstancia:




```
if (condición):
    (bloque)
```

Para modicar la ejecución lineal de las instrucciones, los lenguajes de programación cuentan con instrucciones denominadas estructuras de control. La estructura de control if ... permite que un programa ejecute unas instrucciones cuando se cumplan una condición

```
numero = int(input("Escriba un número positivo: "))
if numero < 0:
    print("Parece que no me has entendido")
    print("¡Le he dicho que escriba un número positivo!")
print("Ha escrito el número", numero)
print("Adios")
```

- La primera línea contiene la condición a evaluar y es una expresión lógica y, esta debe terminar (:).
- El bloque de órdenes que se ejecutan cuando la condición se cumple (es decir, cuando la condición es el boleano TRUE).
- Este bloque debe ir sangrado, utiliza el sangrado para reconocer las líneas que forman un bloque de instrucciones.

Al escribir dos puntos (:) al final de una línea, IDLE sangrará automáticamente las líneas siguientes. Para terminar un bloque, basta con volver al principio de la línea.

En muchos lenguajes de programación, la norma consiste en encerrar las instrucciones a agrupar mediante llaves o paréntesis. Además, para facilitar la lectura del programa (por parte del hombre, que no de la máquina), suele recurrirse a sangrar los bloques.

Una característica de Python es que hace de esa costumbre (el sangrado *indentation* en ingles) la norma y la pertenencia de una línea a un bloque se determina según su margen: no existen delimitadores para los bloques.

- **Tipo booleanos.** Dos valores: TRUE (verdadero) y FALSE (falso). Unas operaciones (entre otras) que dan como resultado booleanos son las comparaciones (< menor que; == igual que ; ! = distinto de)

```
>>> 4 == 5
False
>>> 4 < 5
True
>>> type("hola" != "adiós")
<class 'bool'>
```

PYTHON utiliza los enteros 0 y 1 para representar estos valores, dando lugar a situaciones como esta:

```
>>> True + True
2
>>> type(True * False)
<class 'int'>
```

- **Operadores lógicos: not, and, or**

A	B	A and B	A or B
V	V	V	V
V	F	F	V
F	V	F	V
F	F	F	F

A	not A
V	F
F	V

El orden de prioridad de los operadores es: **not**, **and** y **or**. En caso de querer que las operaciones se realicen en otro orden, se pueden utilizar paréntesis.

Leyes de De Morgan.— Si la condición para poder salir a jugar es `lentejas and deberes`; la condición contraria, que determina cuándo no se puede salir, es `not (lentejas and deberes)`.

```
 >>> True and True
 True
 >>> True or False
 True
 >>> False or False
 False
 >>> not False  and False
False
>>> not(False and False)
True
>>> (True or False) and not(False and True)
True
>>> True or False and False
True
>>> (True or  False) and False
False
```

Cualquier objeto puede ser evaluado para valor booleano. Los siguientes valores son falsos:

None, False, 0, 0.0, 0j, ”,(),[],{}

Todos los demás valores se consideran verdaderos, por lo que los objetos de muchos tipos siempre son verdaderos.

OPERACIÓN	RESULTADO	NOTA
A or B	Si A es falso, entonces B. En otro caso A	(1)
A and B	Si A es falso, entonces A. En otro caso B	(2)
not A	Si A es falso, entonces True. En otro caso False	(3)

- (1) Solamente se evalúa el segundo argumento si el primero es falso.
- (2) Solamente se evalúa el segundo argumento si el primero es verdadero
- (3) **not**, tiene una prioridad más baja que los operadores no booleanos. Por ejemplo, **not A == B** es evaluado como **not (A == B)** y **A == not B** produce un error.

- **Operadores de comparación.** PYTHON permite utilizar al menos 8 operadores de comparación, ($==, !=, <, <=, >, >=$is, is not) a casi todas clases de objetos, e incluso entre objetos de clases distintas (devolviendo un resultado poco significativo, en general.

```
>>> 5 < 6
True
>>> 5 <= 6
True
>>> 'pedro' < 'marta'        orden alfabético
False
3 is 3.0                     ¿identicos?
False
```

```
>>> 5 == 6                   ¿iguales?
False
>>> 'pedro' != 'marta'       ¿distintos?
True
>>> 1 > 2 != 5
True
 3 is not 3                  ¿no identicos?
 False
```

  - Todos tienen la misma prioridad (que es más alta que la de las operadores booleanos).
  - Las comparaciones pueden ser encadenadas arbitrariamente; Por ejemplo, ($\mathbf{x} < \mathbf{y} <= \mathbf{z}$) es equivalente a ($\mathbf{x} < \mathbf{y}$ *and* $\mathbf{y} <= \mathbf{z}$)

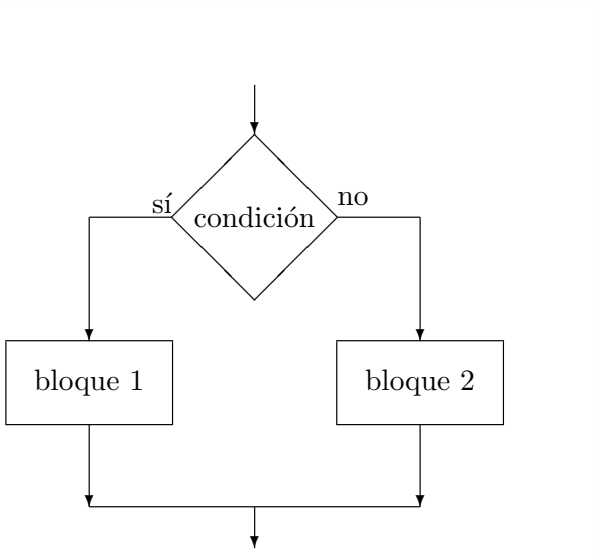
```
>>> 1 < 2 < 5 >= 4 != 3
True
>>> 1 < 2 > 5 >= 4 != 3
False
>>> 1 < 2 > 5 or 4 != 3
True
 >>> "clase" != "murmullo" or "suspenso"
True
```

  Es importante distinguir entre los operadores $=$, $==$ y **is** que hemos visto:

  - $=$ sirve para asignar un valor a una variable. No es simétrico: a la izquierda se pone el nombre de la variable y a derecha, el valor que toma.
  - $==$ compara dos objetos y devuelve un valor «bool» (True o False).
  - **is** compara si dos objetos son idénticos y devuelve un valor «bool» (True o False).

```
>>> v = 2
>>> v == 3
False
>>> v = 3
>>> 3.0 == v
True
>>> v is 3.0
False
```

- **Estructura de control** IF ... ELSE ... Con esta estructura podemos que el programa ejecute unas instrucciones cuando se verifica una condición y otras instrucciones cuando no.




```
if (condición):
    | (bloque 1)
else:
    | (bloque 2)
```

```
edad = int(input("¿Cuántos años tiene ? "))
if edad < 18:
    print("Es usted menor de edad")
else:
    print("Es usted mayor de edad")
print("¡Hasta la próxima!")
```

EJERCICIOS

---

1. Escriba un programa que pida dos números y que conteste cuál es el menor y cuál el mayor o que escriba que son iguales.
2. Escribe un programa que resuelva ecuaciones lineales y se comporte como el del ejemplo:

```
$ python ec_lineal.py
Programa para resolver una
ecuación del tipo ax+b=0

Introduzca los parámetros.
a = 3
b = 5

La solución es -1.67.
```

```
$ python ec_lineal.py
Programa para resolver una
ecuación del tipo ax+b=0

Introduzca los parámetros.
a = 0
b = 5

La ecuación no tiene solución.
```

```
$ python ec_lineal.py
Programa para resolver una
ecuación del tipo ax+b=0

Introduzca los parámetros.
a = 0
b = 0

Cualquier número es soluc.
```

Ayuda,

```
>>> round(-1.6666666666666667, 2)
 1.67
```

3. Escribe un programa que resuelva ecuaciones cuadráticas y se comporte como el del ejemplo:

```
$ python ec_cuad.py
Programa para resolver una
ecuación del tipo ax**2+bx+c=0

Introduzca los parámetros.
a = 2
b = 7
c = 3

Dos soluciones: -0.5 y -3.0
```

```
$ python ec_cuad.py
Programa para resolver una
ecuación del tipo ax**2+bx+c=0

Introduzca los parámetros.
a = 1
b = 2
c = 1

Una solución: -1.0
```

```
$ python ec_cuad.py
Programa para resolver una
ecuación del tipo ax**2+bx+c=0
Introduzca los parámetros.
a = 1
b = 0
c = 1

No hay soluciones reales.
```

4. ¿Qué resultados se obtendrán al evaluar las siguiente expresiones? Calcula primero a mano el valor resultante de cada expresión y comprueba, con la ayuda del ordenador, si tu resultado es correcto.

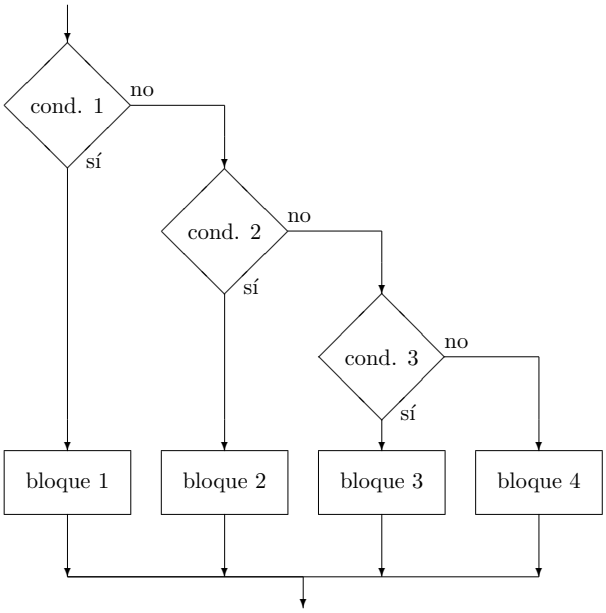
```
>>> 5 > 4
>>>4 <= 4
>>>"torre" > "casa"
>>>5 != 4
>>> True == True != 1
>>> True == True != False
>>> 5 <= 10 < 45
>>> False >= 0
>>> 3 < 5 >= 5
>>> type(True + 2)
>>> (1 < 2 < 5) and (12 < 23)
>>> 1 or abcd
>>> 0 or abcd
>>> (1 != 2 < 5) and (10 < 23)
>>> "silencio" != "amor" or "aprobado"
>>> "clase" != "murmullo" or "suspenso"
>>> "silencio" != "tranquilidad" and "suspenso"
>>> "fresno" <= "santander"
>>> 0 and abcd
>>> 1 and abcd
>>> print(2**0 * False + 2**2**True * False+2**True,"Torre","la", False)
>>> not(True) or False
>>> not(not(True))
```

5. Queremos comprar un regalo de una lista de bodas y elegir el más caro que permita nuestro presupuesto. Así, nuestro programa preguntará al usuario cuánto dinero está dispuesto a gastar y le informará del regalo más costoso que puede adquirir, dentro de una lista como esta:

   vehículo (30 000), vajilla (600), TV (400), abrebotellas (1)

   La sentencia condicional IF *admite una estructura mas compleja que la dual* IF/ELSE.

   Podemos plantear este programa siguiendo un esquema como el siguiente: si se cumple una condición (tener un presupuesto suficiente para el objeto más caro), llevamos a cabo una acción (comprarlo); en otro caso, pasamos a comprobar otra condición; y así sucesivamente. Si una de las condiciones se cumple, se ejecuta el bloque correspondiente y no se evalúa el resto de las condiciones. Escribe un programa que siga este esquema utilizando `if` *y* `else` *(sin atender al código que aparece a la derecha). Pueden incluirse tantas sentencias* ELIF *como se necesite: se ejecuta únicamente el bloque correspondiente a la primera condición que se verifica, o el correspondiente a la instrucción* ELSE *(en caso de estar presente), si no se verifica ninguna.*



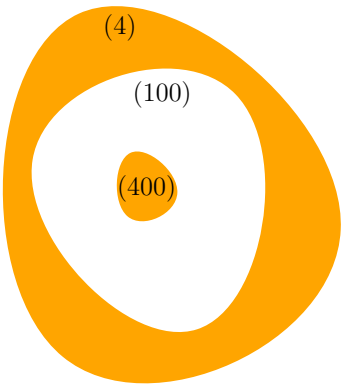

```
if (cond. 1):
    | (bloque 1)
elif (cond. 2):
    | (bloque 2)
elif (cond. 3):
    | (bloque 3)
else:
    | (bloque 4)
```

Las bifurcaciones que obedecen a un esquema como el anterior pueden programarse (según aparece a la derecha) mediante una forma más de la sentencia `if`, que utiliza la palabra reservada `elif`. Escribe el programa que acabamos de hacer utilizando `elif`.

6. Escribe un programa para calcular la nota alfabética de una nota numérica leída por teclado. El programa debe decir si la nota es incorrecta o si es suspenso, aprobado, notable, sobresaliente o matrícula de honor.

7. Diseña un programa para calcular en qué cuadrante esta un punto del plano bidimensional, leído por teclado. El programa debe decir si el punto está en el primer, segundo, tercer, cuarto cuadrante o si está en el eje X, eje Y o si es el origen de coordenadas.

8. **Años bisiestos.**— En el calendario juliano, todos los años tenían 365 días, salvo uno de cada cuatro, que tenía 366. Con esta cuenta se produce un desfase con respecto al año solar de aproximadamente tres días de adelanto cada cuatrocientos años. Por esto motivo, en la reforma gregoriana del calendario se decidió que, de cada cuatro años finiseculares, solo uno fuera bisiesto. Así, el año 2 000 fue bisiesto, pero no el 1 900, ni lo será el 2 100.



bis_1.py

```
1  print ('Escriba un año:')
2  a=int(input())
3
4  if a % 400 == 0:
5      print('bisiesto')
6  if a %100 == 0:
7      print('normal')
8  if a % 4 == 0:
9      print('bisiesto')
10 else:
11     print('normal')
```

bis_2.py

```
1  print('Escriba un año:')
2  a=int(input())
3
4  if a % 400 == 0:
5      print('bisiesto')
6  elif a % 100 == 0:
7      print('normal')
8  elif a % 4 == 0:
9      print ('bisiesto')
10 else:
11     print('normal')
```

Comprueba si alguna de las propuestas `bis_1.py` y `bis_2.py` determina correctamente si un año es bisiesto o no. Para ello, habrá que considerar cuatro «tipos» de años, según la región del dibujo a la que pertenecen: los múltiplos de 400; los que, no siéndolo, sí son múltiplos de 100; los que son múltiplos de 4 pero no son finiseculares; y, por último, los que no son múltiplos de 4. Las regiones coloreadas se corresponden con los años bisiestos.

Representa en forma de diagrama de flujo los dos algoritmos propuestos. Escribe `bis_2.py` sin utilizar `elif`, sino solamente `if` y `else`.

El programa `bis_3.py` discrimina los años bisiestos de los ordinarios utilizando una única bifurcación condicional:

bis_3.py

```
print('Escriba un año:')
a=int(input())

if a %400 == 0 or not a % 100 == 0 and a % 4 == 0:
    print('bisiesto')
else:
    print('normal')
```

bis_4.py

```
print('Escriba un año: ')
a = int(input())

if ##  CONDICIÓN
    print('normal')
else:
    print('bisiesto')
```

Completa el programa `bis_4.py` escribiendo la condición contraria a la que se utiliza en `bis_3.py` y sin utilizar el operador `not`.

9. Queremos vallar una finca rectangular. Escribe un programa que pregunte al usuario tres datos: las dimensiones de la finca (en metros) y los metros de valla de los que dispone. El programa dirá después si el usuario tiene suficiente material o no.

```
$ python valla.py
Lados del rectángulo:
10
100
Valla disponible: 200
Vaya a por más valla.
```

```
$ python valla.py
Lados del rectángulo:
10
100
Valla disponible: 220
Puede cerrar la finca.
```

10. Escribe un programa para convertir de una serie de unidades de superficie a metros cuadrados.

```
$ python superficie.py
¿Cuánto mide la finca? 10
1.| Hectáreas
2.| Áreas
3.| Fanegas
4.| Acres
¿En qué unidades lo ha escrito? 4
Eso son 40469 m.
```

```
$ python superficie.py
¿Cuánto mide la finca? .5
1.| Hectáreas
2.| Áreas
3.| Fanegas
4.| Acres
¿En qué unidades lo ha escrito? 1
Eso son 5000 m.
```

11. Ahora queremos clasificar triángulos a partir de la longitud de sus lados:

```
$ python clasifica.py
Introduzca el primer lado: 7
Introduzca el segundo lado: 7
Introduzca el tercer lado: 7
Un triángulo acutángulo y equilátero
$ python clasifica.py
Introduzca el primer lado: 7
Introduzca el segundo lado: 7
Introduzca el tercer lado: 6
Un triángulo acutángulo e isósceles
$ python clasifica.py
Introduzca el primer lado: 12
Introduzca el segundo lado: 5
Introduzca el tercer lado: 13
Un triángulo rectángulo y escaleno
$ python clasifica.py
Introduzca el primer lado: 2.2
Introduzca el segundo lado: 1.4
Introduzca el tercer lado: 1
Un triángulo obtusángulo y escaleno
```

```
$ python clasifica.py
Introduzca el primer lado: 5
Introduzca el segundo lado: 10
Introduzca el tercer lado: 5
Esos lados no definen un triángulo.
```

Recomendaciones:
Este programa tiene que hacer dos cálculos distintos: decidir, por una parte, si es acutángulo, rectángulo u obtusángulo; y, por otra, si es equilátero, isósceles o escaleno. Será más sencillo hacer los dos cálculos por separado.
Para clasificar el triángulo en función de los ángulos podemos recurrir a la fórmula

$$a^2 = b^2 + c^2 \qquad \text{(donde } a \text{ es el lado mayor)},$$

que se cumple cuando el triángulo es rectángulo. En caso contrario, viendo cuál de los términos es mayor se deduce si el triángulo es acutángulo u obtusángulo.

12. **Escapes.** Las cadenas que hemos visto consistían en sucesiones de caracteres "normales": letras, dígitos, signos de puntuación, etc. La barra invertida `\` se denomina carácter de escape, indica que el siguiente carácter tiene un significado especial. Si el carácter que le sigue es la letra n, `\n` se interpreta como un salto de línea (la n viene del término "new line", es decir, «nueva línea»). Ese par de caracteres forma una secuencia de escape y denota un único carácter. El tabulador esta codificado por `\t`.

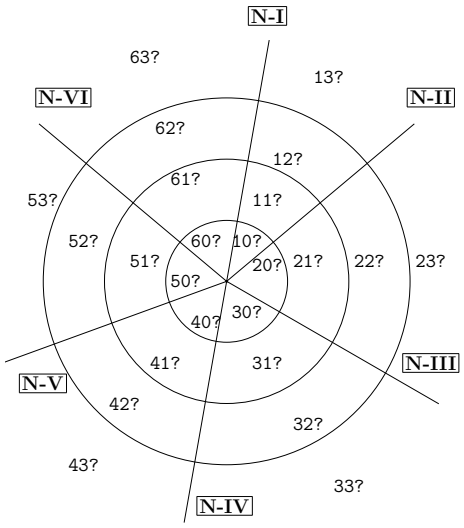
```
>>> print("Hola,\nSantander.")
Hola,
Santander.
>>> print("Hola,\tSantander")
Hola,  Santander
>>> print("Hola,\
 Santander")
Hola, Santander
```

> Si no se quiere que PYTHON añada un salto de línea al final de un PRINT(), se debe añadir al final el argumento `end= ' ',`

```
>>> print("hola");  print("adios")
hola
adios
>>> print("hola", end='');  print("adios")
holaadios
>>> print("hola", end=',');  print("adios")
hola,adios
```

13. **Coordenadas polares.**— Haz un programa para localizar el origen de las carreteras nacionales de la península, tomando en consideración los dos primeros dígitos del código de la carretera.

```
$ python nacional.py
Carretera N- 611
Arranca de un punto localizado entre las radiales N-6 y N-1,
a una distancia de Madrid de entre 100 y 200 km.
$ python nacional.py
Carretera N- 120
Arranca de un punto localizado entre las radiales N-1 y N-2,
a una distancia de Madrid de entre 200 y 300 km.
$ python nacional.py
Carretera N- 232
Arranca de un punto localizado entre las radiales N-2 y N-3,
a una distancia de Madrid de entre 300 y 400 km.
$ python nacional.py
Carretera N- 634
Arranca de un punto localizado entre las radiales N-6 y N-1,
a una distancia de Madrid de entre 300 y 400 km.
```



> Formateo de cadenas. Las llaves y caracteres dentro de las mismas (llamados campos de formato) son reemplazadas con los objetos pasados en el método `str.format(),`. Un número en las llaves se refiere a la posición del objeto pasado en el método.

```
>>> a = 6; b = a + 1; r = 1
>>> print('Arranca de un punto localizado entre las radiales N-{0} y N-{1},\na una distancia de \
... Madrid de entre {2} y {3} km.'.format(a, b, r*100, (r + 1)*100))
Arranca de un punto localizado entre las radiales N-6 y N-7,
a una distancia de Madrid de entre 100 y 200 km.
```

Atendiendo a la paridad del tercer dígito, puedes completar el programa calificando la carretera como «radial» o «transversal».

14. Queremos mandar una carta (un sobre o caja sin forma de tubo; no consideramos las tarjetas postales) a un punto de España. Escribe un programa que pida al usuario, además del peso, las tres dimensiones del objeto que vamos a mandar (largo, ancho y alto), determine si podemos enviarlo como carta y calcule la tarifa.

`http://www.correos.es/ss/Satellite/site/aplicacion-1349167812806-herramientasyapps/detalleapp-sidioma=es`